

7056-0200/P5943/SPL

CERTIFICATE OF MAILING 37 CFR §1.10

"Express Mail" Mailing Label Number: EL 782719285 US

Date of Deposit: October 12, 2001

I hereby certify that this paper, accompanying documents and fee are being deposited with the United States Postal Service "Express Mail Post Office to Addressee" Service under 37 CFR §1.10 on the date indicated above and is addressed to Commissioner for Patents, Box Patent Application, Washington, D.C. 20231.

Jose Ramos

UNITED STATES PATENT APPLICATION

FOR

**METHOD AND APPARATUS FOR
STATEMENT BOUNDARY
DETECTION**

INVENTOR:

DAVID S. ALLISON

PREPARED BY:

**COUDERT BROTHERS LLP
333 SOUTH HOPE STREET
23RD FLOOR
LOS ANGELES, CALIFORNIA 90071**

213-229-2900

BACKGROUND OF THE INVENTION

1. FIELD OF THE INVENTION

5 The present invention relates to the field of computer code parsers, and in particular to a method and apparatus for statement boundary detection.

 Sun, Sun Microsystems, the Sun logo, Solaris and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United
10 States and other countries. All SPARC trademarks are used under license and are trademarks of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

15 2. BACKGROUND ART

 In programming languages, a program is divided into a series of statements, each of which typically execute sequentially. A language parser determines where one statement ends and another begins. Typically, a programmer must insert a special token
20 at the end of the statement. Inserting a special token at the end of each statement is inefficient. This is better understood by a review of programming languages.

Programming Languages

Programming languages are used to express a set of detailed instructions for a digital computer. A programming language consists of characters and rules for combining them into symbols and words.

Many kinds of programming languages have been developed over the years. Initially programmers wrote instructions in machine language. This coded language, which can be understood and executed directly by the computer without conversion or translation, consists of binary digits representing operation codes and memory addresses. Because it is made up of strings of 1s and 0s, machine language is difficult for humans to understand or write. Assembly language was devised for greater convenience. It enabled programmers to express instructions in alphabetic symbols (e.g., AD for add and SUB for subtract) rather than in numbers.

Although assembly language with its mnemonic code was easier to use than machine language, it was clearly desirable to develop programming languages that more closely resembled human communication. The first so-called high-level language was FORTRAN (acronym for Formula Translation), invented in 1956. FORTRAN was well suited to scientists and mathematicians because it was similar to mathematical notations. It did, however, present some difficulty for those in nonmathematically oriented fields. As a result, a more practical programming language known as COBOL (Common Business-Oriented Language) was devised several years later (1960). COBOL employs words and syntax resembling those of ordinary English. Later, other languages even easier to learn and use were introduced. BASIC (Beginner's All-Purpose Symbolic

Instruction Code), for example, can be readily mastered by the layperson and is used extensively in schools, businesses, and homes for microcomputer programming. C is a high-level language that can function as an assembly language; much commercial software is written in this flexible language. Another versatile language widely used for microcomputer as well as minicomputer applications is Pascal (probably named for the French scientist-philosopher Blaise Pascal).

Other high-level programming languages possess unique features that make each one suitable for a specific application. Some examples are APT (Automatically Programmed Tools), for numerical control of industrial machine tools, and GPSS (General-Purpose Simulation System), for constructing simulation models. LISP (List Processing) can be used to manipulate symbols and lists rather than numeric data; it is often used in artificial-intelligence applications. Fourth-generation languages (4GLs) are closer to human language than are high-level (or third-generation) languages. They are used primarily for database management or as query languages; examples include FOCUS, SQL (Structured Query Language), and dBASE. Object-oriented programming languages, such as C++ and Smalltalk, write programs incorporating self-contained collections of data structure or computational instructions (called "objects"). New programs can be written by reassembling and manipulating the objects.

Compiler

Typically, program source code is compiled before it can be executed. Figure 1 illustrates a compiler which translates program source code into computer readable bytecode. The compiler 110 comprises a parser 101, a translator 103, and a code

generator 105. The parser 101 receives input in the form of source code 100 and generates a high-level representation 102 of the program code. This high-level representation 102 may include, for example, a list of statements sorted by order of execution and a list of unique variable identifiers.

5

The translator 103 receives the high level representation 102 and translates the operations into a sequential representation (or intermediate form) 104 that describes the program operations. The sequential representation 104 is transformed by code generation process 105 into executable code 106 for a target simulation system. The code generator
10 may implement one or more optimization techniques (e.g., changing the sequence of executed statements).

Statement Syntax

15

A program is divided into a series of statements, each of which typically execute sequentially. The structure of the statements is determined by the syntax of the programming language. When a program is compiled, first, a parser goes through the text of the source code to associate individual characters or strings of characters in the source code with structural parts of the programming language according to the syntax of the
20 language.

For example, a parser for the C programming language would parse the string “x++; calc = x + y;” as follows: “x” is a variable, “++” is an increment operator, “;” indicates the end of a statement, “ “ is ignored, “calc” is a variable, “ “ is ignored, “x” is

a variable, “+” is an addition operator, “ “ is ignored, “y” is a variable, and “;” indicates the end of a statement.

Statement Terminator Tokens

5

The parser must determine where one statement ends and the next statement begins in the input stream containing the source code for the program. This is traditionally accomplished by requiring the programmer to insert a special token at the end of each statement. For the C programming language, the statement end token is a “;”.

- 10 Other programming languages use difference tokens, including a line-feed or carriage return. In some programming languages (e.g., BASIC), the end of a statement is signified by either a carriage return or a special character between two statements on the same line.

SUMMARY OF THE INVENTION

The present invention provides a method and apparatus for statement boundary detection. In one embodiment of the present invention, a parser determines a natural end
5 of a statement, where possible, based upon the context of the input stream and the syntax of the programming language. Thus, no statement terminator is necessary when a natural end to a statement is determined. The parser uses the natural end of a statement to terminate one statement and begin parsing another statement.

10 In one embodiment, a special statement termination token is required to terminate a statement when no natural statement end exists. In another embodiment, a special statement termination token can be used to terminate a statement when a natural end of the statement exists.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features, aspects and advantages of the present invention will become better understood with regard to the following description, appended claims and
5 accompanying drawings where:

Figure 1 is a block diagram of a compiler.

Figure 2 is a block diagram of a partial syntax for a programming language in
10 accordance with one embodiment of the present invention.

Figure 3 is a flow diagram of the process of determining statement divisions in an input stream of source code in accordance with one embodiment of the present invention.

15 Figure 4 is a flow diagram of the process followed in determining statement ends in accordance with one embodiment of the present invention using the syntax of Figure 1 and the input stream of "i x i y x = y + 1 y = 2".

Figure 5 is a flow diagram of the process of determining statement divisions in an
20 input stream of source code in accordance with one embodiment of the present invention.

Figure 6 is a block diagram of a general purpose computer.

DETAILED DESCRIPTION OF THE INVENTION

The invention is a method and apparatus for statement boundary detection. In the following description, numerous specific details are set forth to provide a more
5 thorough description of embodiments of the invention. It is apparent, however, to one skilled in the art, that the invention may be practiced without these specific details. In other instances, well known features have not been described in detail so as not to obscure the invention.

Natural End of a Statement

In certain contexts, the syntax of a programming language will cause there to be a natural end of a statement in an input stream of source code. For example, Figure 2 illustrates a partial syntax for a programming language. Rule one 200 states that a
15 program, P, is composed of a statement, S; white space, W, followed by a statement, S; a statement, S, followed by white space, W; a statement, S, followed by a program, P; or white space, W, followed by a statement, S, followed by a program, P. Rule two 205 states that a statement, S, is an assignment, A; an assignment, A, followed by the statement terminator “;”; an assignment, A, followed by white space, W, followed by the
20 statement terminator “;”; a declaration, D; a declaration, D, followed by the statement terminator “;”; or a declaration, D, followed by white space, W, followed by the statement terminator “;”.

Rule three 210 states that an assignment, A, is a variable, V, followed by an “=”
25 followed by an expression, E; a variable, V, followed by white space, W, followed by an

“=” followed by an expression, E; a variable, V, followed by an “=” followed by white space, W, followed by an expression, E; or a variable, V, followed by white space, W, followed by an “=” followed by white space, W, followed by an expression, E.

- 5 Rule four 215 states that an expression, E, is a number, N; a variable, V; a number, N, followed by an operator, O, followed by a number, N; a number, N, followed by white space, W, followed by an operator, O, followed by a number, N; a number, N, followed by an operator, O, followed by white space, W, followed by a number, N; a number, N, followed by white space, W, followed by an operator, O, followed by white space, W, followed by a number, N; a variable, V, followed by an operator, O, followed by a number, N; a variable, V, followed by white space, W, followed by an operator, O, followed by a number, N; a variable, V, followed by an operator, O, followed by white space, W, followed by a number, N; a variable, V, followed by white space, W, followed by an operator, O, followed by white space, W, followed by a number, N; a number, N, followed by an operator, O, followed by a variable, V; a number, N, followed by white space, W, followed by an operator, O, followed by a variable, V; a number, N, followed by an operator, O, followed by white space, W, followed by a variable, V; a number, N, followed by white space, W, followed by an operator, O, followed by white space, W, followed by a variable, V; a variable, V, followed by an operator, O, followed by a variable, V; a variable, V, followed by white space, W, followed by an operator, O, followed by a variable, V; a variable, V, followed by an operator, O, followed by white space, W, followed by a variable, V; or a variable, V, followed by white space, W, followed by an operator, O, followed by white space, W, followed by a variable, V;.
- 10
- 15
- 20

Rule five 220 states that an operator, O, is a "+", "-", "*", or "/". Rule six 225 states that a declaration, D, is a type, T, followed by white space, W, followed by a variable, V; Rule seven 230 states that a type, T, is an integer, "i", or a character, "c". Rule eight 235 states that a number, N, is a digit, DI, or a "-" followed by a digit, DI.

- 5 Rule nine 240 states that a digit, DI, is "0"; "0" followed by a digit, DI; "1"; "1" followed by a digit, DI; "2"; "2" followed by a digit, DI; "3"; "3" followed by a digit, DI; "4"; "4" followed by a digit, DI; "5"; "5" followed by a digit, DI; "6"; "6" followed by a digit, DI; "7"; "7" followed by a digit, DI; "8"; "8" followed by a digit, DI; "9"; or "9" followed by a digit, DI;

10

Rule ten 245 states that a variable, V, is a non-reserved letter, NRL, or a letter, L, followed by a variable end string, VES. Rule eleven 250 states that a non-reserved letter, NRL, is any letter other than "i" or "c". Rule twelve 255 states that a letter, L, is any letter. Rule thirteen 260 states that a variable end string, VES, is a number, N; a number, N, followed by a variable end string, VES; a letter, L; or a letter, L, followed by a variable end string, VES. Rule fourteen 265 states that white space, W, is a " "; a " " followed by white space, W; a carriage return; or a carriage return followed by white space, W.

15

- 20 Using the rules of Figure 2, the source code input stream of "i x i y x = y + 1 y = 2" contains three natural ends of statements. A ";" is unnecessary between the first "x" and the second "i" because the only way the beginning of the input stream fits the language syntax is if "i x" is the first statement. The statement is a declaration, with the type being integer and the variable being "x". Likewise, a ";" is unnecessary between the first "y" and the second "x" because the only way that portion of the input stream fits the

language syntax is if “i y” is the second statement. The statement is a declaration, with the type being integer and the variable being “y”.

Similarly, a “;” is unnecessary between the “1” and the third “y” because the only way that portion of the input stream fits the language syntax is if “x = y + 1” is the first statement. The statement is an assignment, “x” as the first variable, white space, “=”, white space, and “y + 1” as the expression. The expression has “y” as the variable, white space, “+”, white space, and “1” as the number.

Parsing Using Natural Ends of Statements

In one embodiment of the present invention, a parser determines a natural end of a statement, where possible, based upon the context of the input stream and the syntax of the programming language. Thus, no statement terminator is necessary when a natural end to a statement is determined. The parser uses the natural end of a statement to terminate one statement and begin parsing another statement.

Figure 3 illustrates the process of determining statement divisions in an input stream of source code in accordance with one embodiment of the present invention. At block 300, it is determined whether there is another character in the input stream. If there is no character in the input stream, at block 310, the current statement is complete and parsing is complete. If there is another character in the input stream, at block 320, the parser determines whether it is consistent with the syntax to include the character as part of the current statement.

If it is consistent with the syntax to include the character as part of the current statement, at block 330, the character is included as part of the current statement and the process continues at block 300. If it is not consistent with the syntax to include the character as part of the current statement, at block 340, the current statement is complete.

5 At block 350, the character is made the beginning of a new statement and the process continues at block 300.

Figure 4 illustrates the process followed in determining statement ends in accordance with one embodiment of the present invention using the syntax of Figure 1 and the input stream of "i x i y x = y + 1 y = 2". At block 400, the parser encounters the character "i". At this point, the program could be of the form S, SW, or SP. The "i" could be the type in a declaration or it could be the beginning of a string that makes a variable. At block 403, the parser encounters the character ". At this point, the parser determines that the character "i" must be the type in a declaration. Thus, the next character must be either white space or the beginning of a variable.

At block 406, the parser encounters the character "x". The parser determines that this must be either the first letter of a string of characters that make up a variable or, since "x" is not a reserved letter, it could be the entire variable. At block 409, the parser encounters the character ". At this point, it is determined from the syntax that the "x" was the entire variable in the declaration statement. Additionally, the first statement is either of the form D or DW;

At block 412, the parser encounters the character "i". At this point, it is determined that "i x" is the first statement of the program. Also, the program is of the

form SP. The P portion of the context expansion is either of the form WS or WSP, where W is the “ “ encountered at block 409. No special token is required for the parser to know where the natural end of the first statement is located.

5 The “i” could be the type in a declaration or it could be the beginning of a string that makes a variable. At block 415, the parser encounters the character “ “. At this point, the parser determines that the character “i” must be the type in a declaration. Thus, the next character must be either white space or the beginning of a variable.

10 At block 418, the parser encounters the character “y”. The parser determines that this must be either the first letter of a string of characters that make up a variable or, since “y” is not a reserved letter, it could be the entire variable. At block 421, the parser encounters the character “ “. At this point, it is determined from the syntax that the “y” was the entire variable in the declaration statement. Additionally, the second statement is
15 either of the form D or DW;

 At block 424, the parser encounters the character “x”. At this point, it is determined that “i y” is the second statement of the program. Also, the program is of the form SWSP. The P portion of the context expansion is either of the form WS or WSP,
20 where W is the “ “ encountered at block 421. No special token is required for the parser to know where the natural end of the second statement is located. At this point, since “x” is neither “i” nor “c”, it is determined that the next statement must be an assignment. Thus, “x” is either the entire variable or the beginning of a string that composes a variable.

25

At block 427, the parser encounters the character “ “. Thus, at this point, it is determined that the “x” is the entire variable and the next character should be either white space or an “=”. At block 430, the parser encounters the character “=”. At block 433, the parser encounters the character “ “. At block 436, the parser encounters the character “y”.

5 At this point, the expression is of the form V, VON, VOV, VWON, VOWN, VWOWN, VWOV, VOWV, or VWOWV.

At block 439, the parser encounters the character “ “. Thus, the expression is of the form V, VWON, VWOWN, VWOV, or VWOWV. At block 442, the parser encounters the character “+”. Now, the parser determines that the expression is not of the form V. At block 445, the parser encounters the character “ “. At this point, the parser determines that the expression is of the form VWOWN or VWOWV. At block 448, the parser encounters the character “1”. At this point, the parser determines that the expression is of the form VWOWN. The “1” could be the entire number or the beginning of a string of numbers.

At block 451, the parser encounters the character “ “. Thus, the expression is “y + 1”. At this point, only white space or a “;” can be included as part of the third statement. At block 454, the parser encounters the character “y”. Thus, at this point the parser determines that the third statement is “x = y + 1”. Also, the program is of the form SWSWSP. The P portion of the context expansion is either of the form WS or WSP, where W is the “ ” encountered at block 451. No special token is required for the parser to know where the natural end of the third statement is located.

At this point, since “y” is neither “i” nor “c”, it is determined that the next statement must be an assignment. Thus, “y” is either the entire variable or the beginning of a string that composes a variable.

5 At block 457, the parser encounters the character “ “. Thus, at this point, it is determined that the “y” is the entire variable and the next character should be either white space or an “=”. At block 460, the parser encounters the character “=”. At block 463, the parser encounters the character “ “. At block 466, the parser encounters the character “2”. At this point, the parser determines that the expression is of the form N. The “2” could
10 be the entire number or it could be the beginning of a string of numbers that compose the number. At block 469, the parser encounters the end of the input stream. Thus, it is determined that the 2 is the entire number and that the last statement is “y = 2”. Also, the program is of the form SWSWSWS, where last W is the “ ” encountered at block 463. No special token is required for the parser to know where the natural end of the last
15 statement is located.

 In one embodiment, a special statement termination token is required to terminate a statement when no natural statement end exists. In another embodiment, a special statement termination token can be used to terminate a statement when a natural end of
20 the statement exists. For example, in one programming language, a statement of the form “variable = variable variable = variable” is permissible in addition to statements of the form “variable = variable. Thus, the statement “x = y z = q” is ambiguous. The input string could be one statement, or the input string could be the statement “x = y” followed by the statement “z = q”. Thus, there is no natural statement end if the programmer
25 wishes the input stream to be two statements. In this instance, if the programmer wishes

the input stream to be two statements, the programmer is required to use a statement terminator to make the statement end explicit. If the statement terminator is a “;”, the correct input stream is “x = y; z = q”.

5 Figure 5 illustrates the process of determining statement divisions in an input stream of source code in accordance with one embodiment of the present invention. At block 500, it is determined whether there is another character in the input stream. If there is no character in the input stream, at block 510, the current statement is complete and parsing is complete. If there is another character in the input stream, at block 520, the
10 parser determines whether the character is a statement terminator. If the character is a statement terminator, at block 530, the current statement is complete, a new statement is begun and the process repeats at block 500.

 If the character is not a statement terminator, at block 540, it is determined
15 whether it is consistent with the syntax to include the character as part of the current statement. If it is consistent with the syntax to include the character as part of the current statement, at block 550, the character is included as part of the current statement and the process continues at block 500. If it is not consistent with the syntax to include the character as part of the current statement, at block 560, the current statement is complete.
20 At block 570, the character is made the beginning of a new statement and the process continues at block 500.

 In one embodiment, the parser uses knowledge gained during parsing in addition to syntactical information to determine natural statement ends. For example, in a
25 language where every variable must be declared before it is used, statements of the form

“variable = variable = variable” are allowed, statements of the form “variable = variable” are allowed and only the variables “x”, “y” and “z” have been declared in the input stream encountered so far, the statement “x=yx=z” is not ambiguous. Since the parser knows that “yx” is not yet a declared variable, the input stream cannot be a single statement of the form “variable = variable = variable”. Instead, the input stream must be the statement “x=y” followed by the statement “x=z”.

Embodiment of Computer Execution Environment (Hardware)

An embodiment of the invention can be implemented as computer software in the form of computer readable program code executed in a general purpose computing environment such as environment 600 illustrated in Figure 6, or in the form of bytecode class files executable within a Java™ run time environment running in such an environment, or in the form of bytecodes running on a processor (or devices enabled to process bytecodes) existing in a distributed environment (e.g., one or more processors on a network), or in the form of bytecodes running on a PDA. A keyboard 610 and mouse 611 are coupled to a system bus 618. The keyboard and mouse are for introducing user input to the computer system and communicating that user input to central processing unit (CPU) 613. Other suitable input devices, a touch-sensitive display for example, may be used in addition to, or in place of, the mouse 611 and keyboard 610. I/O (input/output) unit 619 coupled to bi-directional system bus 618 represents such I/O elements as a printer, A/V (audio/video) I/O, etc.

Computer 601 may include a communication interface 620 coupled to bus 618.

Communication interface 620 provides a two-way data communication coupling via a

network link 621 to a local network 622. For example, if communication interface 620 is an integrated services digital network (ISDN) card or a modem, communication interface 620 provides a data communication connection to the corresponding type of telephone line, which comprises part of network link 621. If communication interface 620 is a local
5 area network (LAN) card, communication interface 620 provides a data communication connection via network link 621 to a compatible LAN. Wireless links are also possible. In any such implementation, communication interface 620 sends and receives electrical, electromagnetic or optical signals which carry digital data streams representing various types of information.

10 Network link 621 typically provides data communication through one or more networks to other data devices. For example, network link 621 may provide a connection through local network 622 to local server computer 623 or to data equipment operated by ISP 624. ISP 624 in turn provides data communication services through the world wide
15 packet data communication network now commonly referred to as the "Internet" 625. Local network 622 and Internet 625 both use electrical, electromagnetic or optical signals which carry digital data streams. The signals through the various networks and the signals on network link 621 and through communication interface 620, which carry the digital data to and from computer 600, are exemplary forms of carrier waves transporting
20 the information.

Processor 613 may reside wholly on client computer 601 or wholly on server 626 or processor 613 may have its computational power distributed between computer 601 and server 626. Server 626 symbolically is represented in Figure 6 as one unit, but server
25 626 can also be distributed between multiple "tiers". In one embodiment, server 626

comprises a middle and back tier where application logic executes in the middle tier and persistent data is obtained in the back tier. In the case where processor 613 resides wholly on server 626, the results of the computations performed by processor 613 are transmitted to computer 601 via Internet 625, Internet Service Provider (ISP) 624, local network 622 and communication interface 620. In this way, computer 601 is able to display the results of the computation to a user in the form of output.

Computer 601 includes a video memory 614, main memory 615 and mass storage 612, all coupled to bi-directional system bus 618 along with keyboard 610, mouse 611 and processor 613. As with processor 613, in various computing environments, main memory 615 and mass storage 612, can reside wholly on server 626 or computer 601, or they may be distributed between the two. Examples of systems where processor 613, main memory 615, and mass storage 612 are distributed between computer 601 and server 626 include the thin-client computing architecture developed by Sun Microsystems, Inc., the palm pilot computing device and other personal digital assistants, Internet ready cellular phones and other Internet computing devices, and in platform independent computing environments, such as those which utilize the Java technologies also developed by Sun Microsystems, Inc.

The mass storage 612 may include both fixed and removable media, such as magnetic, optical or magnetic optical storage systems or any other available mass storage technology. Bus 618 may contain, for example, thirty-two address lines for addressing video memory 614 or main memory 615. The system bus 618 also includes, for example, a 32-bit data bus for transferring data between and among the components, such as processor 613, main memory 615, video memory 614 and mass storage 612.

Alternatively, multiplex data/address lines may be used instead of separate data and address lines.

In one embodiment of the invention, the processor 613 is a SPARC

5 microprocessor from Sun Microsystems, Inc., a microprocessor manufactured by Motorola, such as the 680X0 processor, a microprocessor manufactured for use in a PDA, or a microprocessor manufactured by Intel, such as the 80X86 or Pentium processor. However, any other suitable microprocessor or microcomputer may be utilized. Main memory 615 is comprised of dynamic random access memory (DRAM), and bytecodes
10 for one embodiment of the invention is stored in a portion 627 of main memory 615 during program execution. Video memory 614 is a dual-ported video random access memory. One port of the video memory 614 is coupled to video amplifier 616. The video amplifier 616 is used to drive the cathode ray tube (CRT) raster monitor 617. Video amplifier 616 is well known in the art and may be implemented by any suitable
15 apparatus. This circuitry converts pixel data stored in video memory 614 to a raster signal suitable for use by monitor 617. Monitor 617 is a type of monitor suitable for displaying graphic images.

Computer 601 can send messages and receive data, including program code,
20 through the network(s), network link 621, and communication interface 620. In the Internet example, remote server computer 626 might transmit a requested code for an application program through Internet 625, ISP 624, local network 622 and communication interface 620. The received code may be executed by processor 613 as it is received, and/or stored in mass storage 612, or other non-volatile storage for later
25 execution. In this manner, computer 600 may obtain application code in the form of a

carrier wave. Alternatively, remote server computer 626 may execute applications using processor 613, and utilize mass storage 612, and/or video memory 615. The results of the execution at server 626 are then transmitted through Internet 625, ISP 624, local network 622 and communication interface 620. In this example, computer 601 performs only
5 input and output functions.

Application code may be embodied in any form of computer program product. A computer program product comprises a medium configured to store or transport computer readable code, or in which computer readable code may be embedded. Some examples of
10 computer program products are CD-ROM disks, ROM cards, floppy disks, magnetic tapes, computer hard drives, servers on a network, and carrier waves.

The computer systems described above are for purposes of example only. An embodiment of the invention may be implemented in any type of computer system or
15 programming or processing environment.

Thus, a method and apparatus for statement boundary detection is described in conjunction with one or more specific embodiments. The invention is defined by the following claims and their full scope and equivalents.